SIMSCAPE MODELING OF MOTOR-GENERATOR UNIT COMPONENTS FOR HYBRID ELECTRIC VEHICLE

A Thesis
Presented to
The Academic Faculty

by

Yashdeep Narkhede

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
May 2016

SIMSCAPE MODELING OF MOTOR-GENERATOR UNIT COMPONENTS FOR HYBRID ELECTRIC VEHICLE

Approved by:

Professor David Taylor, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Thomas Habetler
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Michael Leamy
School of Mechanical Engineering
Georgia Institute of Technology

Date Approved: 14 April 2016

To my parents.

ACKNOWLEDGEMENTS

First and foremost, I greatly appreciate the support and guidance provided by Dr. Taylor. I would have not completed this project and dissertation without your motivation and valuable ideas. I am highly obliged to have received all your help and no amount of acknowledgment would parallel all your help and support.

I would like to thank my thesis reading committee, Dr. Habetler and Dr. Leamy, for taking the time of their busy schedules for reading through my thesis and for their indispensable and wise remarks.

A special thanks to my parents who gave the me motivation and support during the course of this research. I would also like to thank my friends and colleagues from whom I learned at every step of the project.

A big thank you to the EcoCAR3 organizers and sponsors for their donation and valuable support and information. A special thanks to Ryan Chladny, MathWorks mentor, for helping me with issues faced with Simscape. I would also like to thank the entire Georgia Tech EcoCAR3 team for their help and support for this project.

On a concluding note and with a grateful heart, I would like to thank all the individuals who directly or indirectly contributed towards the completion of this project and dissertation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The thesis introduces the user to programming in Simscape language. A permanent magnet synchronous machine torque control drive system for hybrid electric vehicles has been analyzed, programmed, using Simscape language, and tested in this thesis. The thesis walks the reader through the process of creating custom components in Simscape language explaining details and syntax of the language at every step. Important excerpts of code for all the components designed, created and used in the process are explained in the thesis and the complete code for the same is provided in the Appendix.

CHAPTER I

INTRODUCTION

Electric machine drive systems have become ubiquitous because of improved efficiency, better stability and control. Such systems are widely studied and can be analyzed in numerous ways. These analyses can differ in their sign conventions, choice of transformations, etc. Multiple simulation platforms are available for simulating these systems such as MATLAB, Simulink, Simscape, etc. These platforms sometimes provide built-in models of electric machines and drive control systems. But their analysis may not align with the sign conventions or the choice of transformations preferred by the user, often causing the user to rework their entire analysis to match the provided blocks.

Products from companies other than MathWorks also exist, but MathWorks is one of the sponsors for the EcoCAR3 competition so their products were chosen for use. A system of equations can be simulated in any of the simulation tools mentioned above, but the tools mainly differ in their style of coding and way of presentation. MATLAB is a text based coding language which often requires numerous lines of code to implement control systems. Although it becomes easier to keep track of configuration parameters, variables, etc., the code becomes unidirectional and fails to capture the bidirectional nature of the system discussed here. The user does not get to see the arrangement of components in the system and it is left to their imagination. Simulink takes a block based approach for simulating systems. However, equations programmed with the help of blocks make it difficult to keep track of wires, basic building blocks, etc., and troubleshooting the logic becomes a tedious task. Simscape is a package available in Simulink, which combines the benefits of text

based programming with a block based perspective for physical system modeling. Consistent with physical system modeling, its ports are bidirectional. The system of equations that is simulated resides in text based files and when compiled, generates a block that can include as much complexity as desired and can also be a cross-domain component. The user gets to access the equations being simulated while being in a clear view of components and domain boundaries such as electrical or mechanical domains, etc.

Simscape Foundation Library contains most basic components such as resistors, capacitors, etc. in electrical domain, springs, dampers, etc. in the mechanical domain, and so on. The codes for these components are made available to the user for reference, or for making additions to existing components. However, Simscape does not provide code for complex systems such as permanent magnet synchronous machines or power converters and controllers. It is up to the user to program such complex systems. The approach taken by [1] is slightly similar to this thesis. Rather than using standard library components, they introduce the concept of creating custom components. But the examples discussed are for creating basic components as they explore the possibility of using custom Simscape components in their lectures. System level design using Simscape language is discussed in [2]. This paper explains the process of creating components with the example of a servomotor. [3] makes use of Simscape language to create custom blocks, but the blocks developed are for power systems.

The use of Simscape language to create custom components is emphasized in this thesis due to the multiple advantages of using the language. One of the advantages, discussed earlier, includes the bidirectional nature of the ports and hence the simulation. There is no limit to the amount of complexity that can be incorporated into the component. Hence, the Simscape component can be made as close to the actual physical component as desired by including factors such as efficiency, thermal

management, etc. Systems that require components beyond the Foundation Library components can be created easily with the language. The simulation time can be controlled cleverly by taking advantage of system physics, i.e. instead of creating a power converter that simulates high frequency switching inside the power converter, averaging converters can be programmed. This helps tremendously for increasing the simulation time step. For instance, the simulation discussed in this thesis can run on time step of 0.01 seconds. This larger step size in turn helps in reduction of time required for simulation. The response becomes visibly discrete using time step of 0.01 seconds and hence 0.001 seconds is used here. The solver is able to simulate more than 4 seconds of system operation within one second of real time.

The purpose of this thesis is to introduce the reader to the development of custom blocks in Simscape environment. This is achieved with an example of a torque control drive for a permanent magnet synchronous machine. The approach taken by this thesis is that it begins with analysis and derivation of the equations for individual components of the drive system, i.e. summarizing [4]. Simscape language environment is then introduced, and all the equations derived are then programmed in individual component blocks representing these components. A custom library is constructed that consists of all such custom built component blocks. These blocks are then used in simulations with different torque request waveforms, and results are presented.

CHAPTER II

THEORY OF OPERATION

2.1 System Overview

A motor-generator drive system is typically comprised of a DC power source, a power converter and an electric machine. The power converter is a combination of a power stage and its controller. The electric machine is operated in either speed control mode or torque control mode by the power converter controller. It determines the voltages at the power converter legs with the help of a suitable strategy. One such system is depicted in Figure 1. The resistors shown in Figure 1 have been included to achieve two perhaps non-standard objectives. The purpose of R_1 is to establish a bias current so that regenerative modes of operation can be explored using a power supply source as opposed to a battery source. The purpose of R_2 is to limit the inrush current to the power converter which includes an internal bus capacitor; this current limiting feature is typically only needed at start-up. In this chapter, all the components in such a drive system are studied individually and mathematical equations are derived.

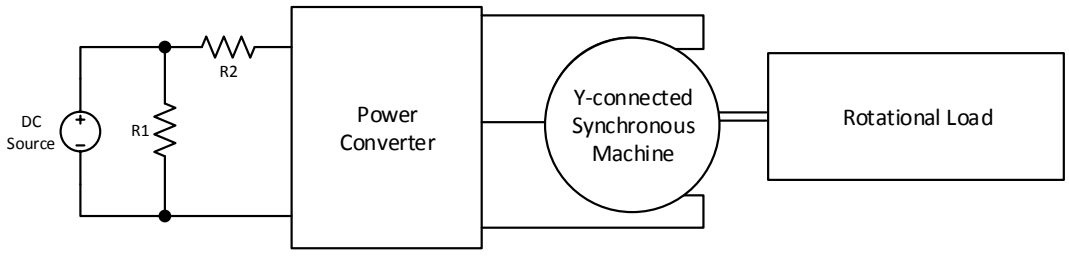

Figure 1: Overview of motor-generator drive system.

2.2 Electric Machine

Figure 2 represents the equivalent circuit of a permanent magnet synchronous machine. The '`a`', '`b`' and '`c`' terminals in the figure are the electrical terminals to which the power converter AC-side terminals are connected. The terminal '`n`' is the electrical reference terminal. The voltages at the terminals '`a`', '`b`' and '`c`' with respect to the voltage at terminal '`n`' represent the across variables, given by v_a, v_b and v_c, and the currents i_a, i_b and i_c represent the through variables for the electrical domain of the machine. E_a, E_b and E_c represent the back-EMFs induced in the stator due to the permanent magnets on the rotor and are given as

$$\begin{bmatrix} E_a \\ E_b \\ E_c \end{bmatrix} = \Lambda_m N\omega \begin{bmatrix} \sin(N\theta) \\ \sin(N\theta - \frac{2\pi}{3}) \\ \sin(N\theta + \frac{2\pi}{3}) \end{bmatrix}$$

The equations defining the machine's electrical dynamics are given by

$$\frac{di_a}{dt} = \frac{1}{L_s}(v_a - R_s i_a + \Lambda_m N\omega \sin(N\theta)) \quad (1)$$

$$\frac{di_b}{dt} = \frac{1}{L_s}(v_b - R_s i_b + \Lambda_m N\omega \sin(N\theta - \tfrac{2\pi}{3})) \quad (2)$$

$$\frac{di_c}{dt} = \frac{1}{L_s}(v_c - R_s i_c + \Lambda_m N\omega \sin(N\theta + \tfrac{2\pi}{3})) \quad (3)$$

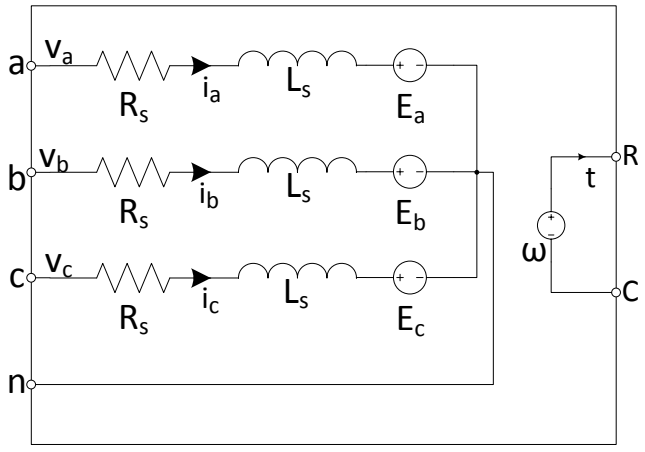

Figure 2: Equivalent circuit of permanent magnet synchronous machine.

where R_s and L_s represent the resistance and inductance per phase, N represents the number of pole pairs, Λ_m represents the permanent magnet flux, ω represents the angular velocity of the rotor and θ represents the angular position of the rotor measured from A-phase magnetic axis to d-axis.

The analysis of AC machines is complicated due to the presence of periodically varying voltages and currents. A change of coordinates is therefore used to transform the machine variables to rotor reference frame. This is a mapping of a three phase system onto the two axes, d-axis and q-axis, that rotate with the rotor. The d-axis is aligned with the permanent magnet flux and the q-axis is orthogonal to the d-axis as shown in Figure 3. The zero axis voltage and current are negligible for a balanced three phase system. The transformation matrices used for the mapping are given by

$$\begin{bmatrix} v_d \\ v_q \\ v_o \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} \cos(N\theta) & \cos(N\theta - \frac{2\pi}{3}) & \cos(N\theta + \frac{2\pi}{3}) \\ -\sin(N\theta) & -\sin(N\theta - \frac{2\pi}{3}) & -\sin(N\theta + \frac{2\pi}{3}) \\ \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} \end{bmatrix} \begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix} \tag{4}$$

$$\begin{bmatrix} i_d \\ i_q \\ i_o \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} \cos(N\theta) & \cos(N\theta - \frac{2\pi}{3}) & \cos(N\theta + \frac{2\pi}{3}) \\ -\sin(N\theta) & -\sin(N\theta - \frac{2\pi}{3}) & -\sin(N\theta + \frac{2\pi}{3}) \\ \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} \tag{5}$$

and

$$\begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} \cos(N\theta) & \cos(N\theta - \frac{2\pi}{3}) & \cos(N\theta + \frac{2\pi}{3}) \\ -\sin(N\theta) & -\sin(N\theta - \frac{2\pi}{3}) & -\sin(N\theta + \frac{2\pi}{3}) \\ \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} \end{bmatrix}' \begin{bmatrix} v_d \\ v_q \\ v_o \end{bmatrix} \tag{6}$$

$$\begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} \cos(N\theta) & \cos(N\theta - \frac{2\pi}{3}) & \cos(N\theta + \frac{2\pi}{3}) \\ -\sin(N\theta) & -\sin(N\theta - \frac{2\pi}{3}) & -\sin(N\theta + \frac{2\pi}{3}) \\ \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} \end{bmatrix}' \begin{bmatrix} i_d \\ i_q \\ i_o \end{bmatrix} \tag{7}$$

This is power invariant form of dq transformation.

Using the transformations defined by (4) through (7), the machine equations are

transformed to the following equations:

$$\frac{di_d}{dt} = \frac{1}{L}(v_d - R_s i_d + N\omega L i_q) \tag{8}$$

$$\frac{di_q}{dt} = \frac{1}{L}(v_q - R_s i_q - N\omega L i_d - \sqrt{\tfrac{3}{2}} N\omega \Lambda_m) \tag{9}$$

The terminals '`R`' and '`C`' of Figure 2 represent the mechanical terminals with angular speed as the across variable and torque as the through variable for the mechanical domain of the machine. The terminal '`R`' can be considered as the rotor and the terminal '`C`' can be considered the case which serves as a reference for the mechanical domain. These terminals are connected to the mechanical rotational load.

The torque produced by the electric machine is evaluated by equating the electrical power input to the rotor and the mechanical power output of the rotor. The electrical power input to the rotor is the power remaining after the copper and iron losses of the stator have been subtracted. The torque equation is therefore given by

$$T = -\Lambda_m N(i_a \sin(N\theta) + i_b \sin(N\theta - \tfrac{2\pi}{3}) + i_c \sin(N\theta + \tfrac{2\pi}{3})) \tag{10}$$

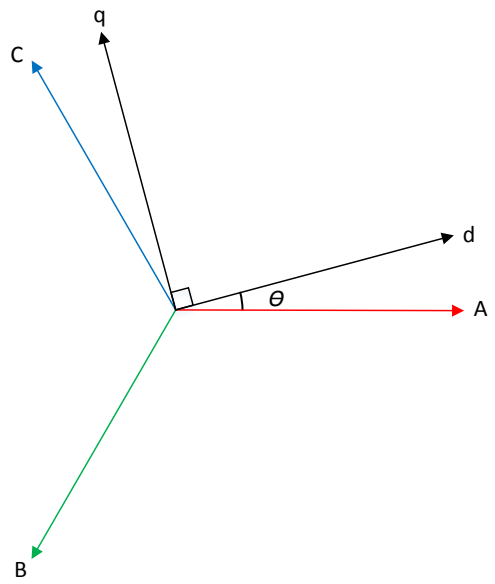

Figure 3: ABC to dq transform.

Using the transformation matrix defined by (5), the torque can be represented in the dq-axis system as

$$T = \sqrt{\tfrac{3}{2}}(N\Lambda_m i_q) \tag{11}$$

The torque of the machine in this case is considered as positive when it flows out of the machine, i.e. the machine is operating as an electric motor.

The rotor inertia and damping coefficient are used to calculate the opposing torque for the electric machine which is given by

$$T_r = J_r\dot{\omega} + B_r\omega \tag{12}$$

where J_r represents the rotor inertia and B_r represents the rotor damping coefficient.

2.3 Power Converter

A power converter transforms DC electrical energy into AC electrical energy and vice versa. The energy transformation is achieved with the help of a power stage and its controller. The power stage consists of the physical devices such as power MOSFETs, IGBTs, etc. These devices are arranged in a three phase bridge and are operated to control the voltages at each leg. Figure 4 shows a three phase bridge power stage along with a DC bus capacitor that is used to maintain a constant DC voltage under fluctuating load.

2.3.1 Power Stage

A power stage for the power converter can be represented as shown in Figure 4. The switches in each leg are operated in complimentary manner to prevent short circuit of the DC side. The voltage at each leg is controlled by PWM signals applied by the power stage control strategy. The PWM duty cycles are dependent on the voltage reference commands received from the power converter controller. Upon averaging, the modulated voltages at the terminals `A`', '`B`' and '`C`' result in the commanded

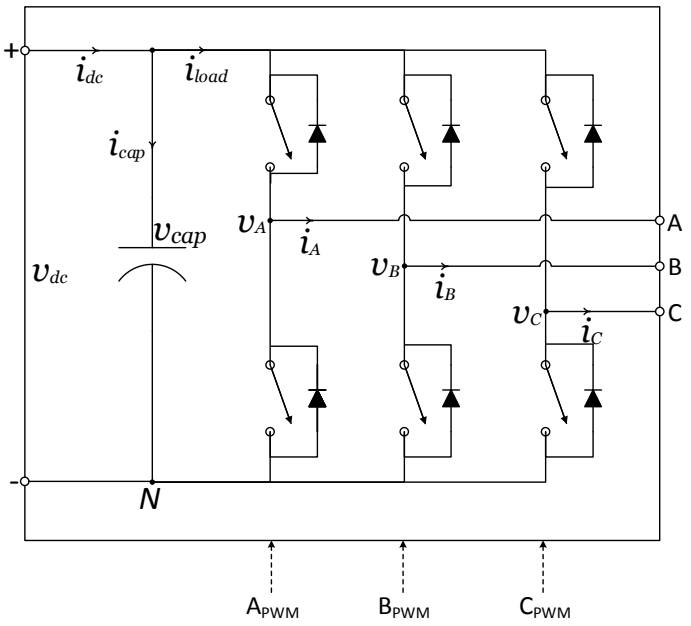

Figure 4: A power stage of a power converter.

voltages. The current through these terminals is dependent on the load connected to power converter.

In Figure 4, i_A, i_B and i_C represent the currents flowing out through the terminals '`A`', '`B`' and '`C`', and v_A, v_B and v_C represent the voltages across the terminals with respect to the terminal '`N`' of the power stage respectively. The capacitor current and voltage are represented by i_{cap} and v_{cap} respectively. The DC side current that enters the power stage bridge is given by i_{load} and is calculated by equating input and output power of the three phase bridge under the assumption that the switches are ideal and lossless. The capacitor current is calculated by using the capacitor dynamic equation. Kirchhoff's Current Law and Kirchhoff's Voltage Law are used to calculate the DC side input current and voltage respectively. The power stage equations are

given by

$$i_{load} = \frac{v_A i_A + v_B i_B + v_C i_C}{v_{dc}} \tag{13}$$

$$i_{cap} = C_{cap} \frac{dv_{cap}}{dt} \tag{14}$$

$$i_{dc} = i_{cap} + i_{load} \tag{15}$$

$$v_{dc} = v_{cap} + i_{cap} R_{ESR} \tag{16}$$

where v_{dc} is the voltage of the DC bus, C_{cap} represents the capacitance and R_{ESR} represents the equivalent series resistance of the capacitor.

2.3.2 Power Converter Controller

The power converter controller provides voltage commands to the power stage based on the torque request, and load current, angular position and angular velocity feedback. Electric machine analysis is used to generate the appropriate reference voltage commands for a given torque request.

The voltages at the power stage AC side terminals are represented as v_A, v_B and v_C and the voltages at the Y-connected machine terminals are represented as v_a, v_b and v_c. The node '`N`' and '`n`' represent the reference nodes for power stage and electric machine respectively. The relation between the power stage and machine terminal voltages is established by using Kirchhoff's Voltage Law as

$$v_A = v_a + v_{nN} \tag{17}$$

$$v_B = v_b + v_{nN} \tag{18}$$

$$v_C = v_c + v_{nN} \tag{19}$$

where, v_{nN} represents the voltage of the node '`n`' with respect to node '`N`'. Also, taking advantage of the balanced three phase system yields:

$$v_a + v_b + v_c = 0 \tag{20}$$

Rearranging these equations to express the voltages experienced by the machine terminals as a result of converter leg voltages is as follows:

$$v_a = \frac{2}{3}v_A - \frac{1}{3}v_B - \frac{1}{3}v_C \quad (21)$$

$$v_b = \frac{2}{3}v_B - \frac{1}{3}v_C - \frac{1}{3}v_A \quad (22)$$

$$v_c = \frac{2}{3}v_C - \frac{1}{3}v_A - \frac{1}{3}v_B \quad (23)$$

Using the relation between the power converter terminal voltages and machine phase voltages given by (21) through (23) and the transformations defined by (4) through (7), a desired set of d-axis and q-axis voltages can be applied to the machine stator by controlling voltages at the power converter output terminals. These voltage commands are generated by using internal current feedback loops.

The torque of the electric machine is proportional to the q-axis current as seen from (11) and can be used to calculate the q-axis reference current value. Based on (11), d-axis current will not cause any difference in the torque produced by the machine, giving infinite possibilities to d-axis reference current selection. To minimize losses in the system, in the constant torque region, the d-axis reference current is set to zero. This strategy is also known as maximum torque per ampere. Since the simulation model developed in this thesis was intended to represent a bench test system that does not operate beyond base speed, consideration of flux weakening was unnecessary. And hence, the reference currents can be represented as

$$i_{q,ref} = \sqrt{\frac{2}{3}}\left(\frac{T_{ref}}{N\Lambda_m}\right) \quad (24)$$

$$i_{d,ref} = 0 \quad (25)$$

where T_{ref} is the torque request.

Using the stator current and angular position feedback, the d-axis and q-axis currents are calculated from the relation defined in (5). PI regulators acting on errors between actual and reference currents generate the respective axis command voltages,

given by

$$v_{d,ref} = -k_p(i_d - i_{d,ref}) - k_i x_d \tag{26}$$

$$v_{q,ref} = -k_p(i_q - i_{q,ref}) - k_i x_q \tag{27}$$

where x_d and x_q are defined by the equations

$$\dot{x_d} = i_d - i_{d,ref}$$

$$\dot{x_q} = i_q - i_{q,ref}$$

The averaged three phase reference voltages that are given as commands to power stage are computed using (6). The zero axis voltage is assumed to be zero.

2.4 Rotational Load

The rotational load applied to the electric machine shaft is a combination of rotor inertia and damping coefficient added to the load inertia and damping coefficient. The load torque is then given by

$$T = J\dot{\omega} + B\omega \tag{28}$$

where ω is the angular velocity of the machine shaft, J and B are the moment of inertia and damping coefficient of the rotor-load combination. The angular position of the load can be calculated from the kinematics

$$\dot{\theta} = \omega \tag{29}$$

2.5 DC Power Source

A DC power source applies a DC voltage across, and passes DC current through, its load. A few examples of DC power sources are a battery pack, a regulated DC power supply, a grid fed rectifier with bus capacitor, etc. The source considered for our bench test setup is the regulated DC power supply. A practical power source contains

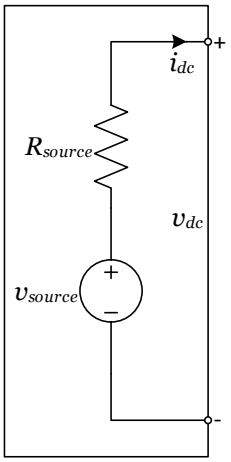

Figure 5: A practical DC power source.

a small parasitic internal resistance. The output voltage of a practical source hence fluctuates with the amount of current drawn from it. This resistance incurs losses by dissipating heat. Figure 5 depicts a practical DC power source. The output voltage equation for such a DC power source is derived from the application of Kirchhoff's Voltage Law and Ohm's Law and is given by

$$V_{dc} = V_{source} - i_{dc}R_{source} \tag{30}$$

From (30), the voltage at the output terminals reduces as the current drawn out of the source increases.

2.6 Resistor

A resistor is a basic component of electrical technology. The resistor follows Ohm's Law and the voltage v_R across it is given as

$$v_R = i_R R \tag{31}$$

where R is the resistance value and i_R is the current through the resistor.

CHAPTER III

SIMSCAPE IMPLEMENTATION

3.1 Simscape Language

Simscape language enables the user to program blocks in cases where the functionality of the Foundation Library components is not sufficient. It is a text based language that allows the user to program mathematical equations and create component blocks. Each component requires its own file and is saved in '`.ssc`' format. The component files are built using a specific command covered in this chapter, and a block is created which can be dragged and used in simulations similar to standard library blocks. This section covers the structure of these component files. The following sections cover the elements of a drive system discussed in the last chapter. The guidelines discussed in this chapter summarizes the content in [5]. An overview of the entire drive system and its operation is explained in the final section of this chapter.

A component file starts with the keyword '`component`' followed by the component name. The component name cannot have any spaces in it. The file name should be the same as the component name. The next line is a comment that gives the block name in the library. The block name can contain spaces. The next few lines of code are optional comments which appear as description in the block dialog box. The component file is then organized to contain the declaration section, followed by the setup, branches and equations sections. Each of the sections, and the component file, concludes with the '`end`' keyword.

With the help of an example, the purpose and contents of each of the sections mentioned above are now explained. Consider the code for a capacitor with equivalent series resistance given in Listing 1. The component name used here is '*MyCapacitor*',

and hence the file name is '`MyCapacitor.ssc`'. The block name is '*Capacitor*'. As mentioned above, the file begins with the keyword '`component`'.

Listing 1: Capacitor code

```
component MyCapacitor
    % Capacitor
    % This block is a capacitor with a user configurable
       value.
    parameters
        C = {0.1,'uF'}; % Capacitance
        ESR = {0.1,'Ohm'}; % equivalent series resistance
        v0 = {0,'V'}; % Initial Voltage
    end
    nodes
        p = foundation.electrical.electrical; % +:top
        n = foundation.electrical.electrical; % -:bottom
    end
    variables
        idc = { 0, 'A' }; % DC-side Current
        vdc = { 0, 'V' }; % DC-side Voltage
        vcap = {value={0,'V'},priority=priority.high}; %
           Capacitor Initial Voltage
    end
    function setup
    if C<=0
        pm_error('simscape:GreaterThanOrEqualToZero','
           Capacitance')
    end
    if ESR<=0
        pm_error('simscape:GreaterThanOrEqualToZero','
           equivalent series resistance')
    end
    vcap = v0;
    end
    branches
        idc   : p.i -> n.i;
    end
    equations
        vdc == p.v - n.v;
        idc == C*vcap.der;
        vdc == vcap + idc*ESR;
    end
end
```

The lines 4 through 17 are the declaration of the component. The section contained in lines 18 through 26 is the setup section followed by the branches section, lines 27 through 29, and the equations section, lines 30 through 34. Line 35 denotes the last line of the component file as marked by the '`end`' keyword.

The declaration section consists of parameters, variables, inputs, outputs and nodes of the components. The parameters are the user configurable attributes of the component such as capacitance, resistance or inductance values, etc. Optional comments right after the parameter declaration, as shown on line 5 through 7 in the code, give the parameter a name in the component dialog box. Each parameter default value is declared as a 'value with unit'. The value can be changed by the user through the dialog box. The unit should be defined in the list of defined units, which can be viewed by typing '`pm_getunits`' in the command window. If the parameter being declared is unitless, '1' should be used instead.

The inputs and outputs sections create physical signal input and output directional ports on the components. Physical connections can carry physical signals such as feedback from machines or sensors, or command signals such as actuator control signals, etc. In default Simscape settings, these connections are represented by thick brown lines. Input and output ports are represented as triangles on the block, input port with tip of the triangle pointing towards the box and output port with tip pointing away from the box. The display name of the port and sides of the block on which these ports are arranged can be defined by an optional comment after the declaration of the port. The syntax for the same is given in Listing 2. Here, the

Listing 2: Example of physical port declaration

```
1     outputs
2         vA_ref = { 0.0, 'V' }; % vA_ref:right
3     end
4     inputs
5         Te_ref = {0, 'N*m'};    % Torque command: left
6     end
```

input is on the left side along with its name. A block can contain a combination of ports and nodes on opposing sides only, i.e. left-right only or top-bottom only. There cannot be any port or node on any adjacent sides. The inputs and outputs are also initialized as 'value with unit'. These are not configurable by the user.

Across and through component variables are declared in the variables section. This section can also contain internal variables that are used in the equations section. The variables are initialized as 'value with unit'. Optional comments give the variables a name. Variables are accessible under the variables tab of the dialog box. Some variables can be hidden from the user. Such variables are called private variables and use the '`access`' keyword to make them private. Priority of variables is used to force the desired conditions on the component. For example, in the capacitor example, line 16 shows that the voltage across the capacitor is assigned as high priority. The solver assigns the desired initial value to the high priority variables first and then satisfies the equations by calculating the values for the low priority variables to start the simulation.

The nodes section is used to define the nodes or the conserving ports of the component. The nodes belong to a physical domain such as electrical, thermal, etc., i.e. nodes in one domain can only connect to other nodes of the same domain. A domain is defined by its across and through variables. If a node is created, it carries the domain across and through variables. The across variable is a measurable quantity across nodes and the through variable is a through quantity which has to balance out. In other words, the sum of through variables at a branch point connected to a node should be equal to zero. The relationship of the component variables to node variables is established using the branches and the equations sections. The lines 9 through 12 describe the node declaration. In this example, two nodes of the electrical domain are declared. The electrical domain is the standard domain defined in the Foundation Library in '`electrical.ssc`' file. The file can be found under

the '`+foundation\+electrical`' directory. This path is therefore used in the node declaration section using the syntax shown in the code on lines 10 and 11. The optional comment after the declaration is used to control the display name of the node and its location on the block.

The lines 18 through 26 are the setup section of the code. This section is used for validating parameters, setting initial conditions and computing derived parameters. Lines 19 through 24 show examples of validating parameters. This part of the setup section is used to ensure all parameters in the component are entered correctly by the user, i.e. this section generates an error if parameters do not satisfy the conditions defined here. In the capacitor example, this feature is used to ensure the value entered for the capacitance is non-negative. Line 25 shows the initial condition setup. Any variables can be set to desired initial values as required by the component and/or simulation. The derived parameter values are calculated based on certain conditions which may be imposed by the simulation. An example could be if a parameter called 'mode' is 1 then the value of a certain other parameter could be 1, else it could be 0.

The branches section is used to define the flow of the through variable in the component, i.e. establish the relation between the component through variables and nodes. In the capacitor component, the through variable '`idc`' flows from node '`p`' to node '`n`' and the syntax for the same is shown on line 28. Since the domain through variables have to balance out, the value of the through variable is subtracted from the branch attached to node '`p`' and is added to the branch attached to the node '`n`'. As the nodes are created using the electrical domain, they carry the 'i' domain through variable. Thus, the relation between the domain through variable and the component through variable is established. For multiple through variables, multiple branches must be defined in this section.

The equations section defines the relation between the across variables and nodes. This section is also used to program the desired behavior of the component. The across

variable '`vdc`' is defined as the voltage between node '`p`' and '`n`' by the equation on line 31. Again, since the nodes are created using the electrical domain, they carry domain across variable '`v`'. The connection between node and component through variable is therefore specified by line 31. For multiple across variables, multiple such equations should be defined. The operator '`==`' represents the bidirectional nature of the equations, e.g. the right hand side can contain variables to be calculated and the left hand side can contain constants so that '$0 == var1 + var2$' is a valid statement. Inputs, outputs, parameters and variables are all related using mathematical expressions in the equations section. As all these entities have units, the units on both sides of the equations should be commensurate. The number of equations should match the number of variables (private and public included).

The '`der`' keyword is used to specify derivatives. For example, the equation

$$i_{dc} = C_{cap}\frac{dv_{dc}}{dt}$$

can be programmed as shown on line 32 using the '`der`' keyword. Similarly, the current simulation time can be used in equations using the '`time`' keyword. If intermediate terms are used in the equations, the '`let-in-end`' construct is used to define these terms as shown in Listing 3. Note the expressions in the let subsection are not bidirectional.

The conditional construct, i.e. the '`if-then-else-end`' construct can also be used in the equations section. There are a few restrictions on using this construct, i.e. each

Listing 3: Example of '`let-in-end`' construct

```
1 % Extract from the PMSM block. Lambda is used as an
    intermediate term.
2 equations
3   let
4         Lambda = sqrt(3/2)*Lambda_m;
5     in
6         T == N*Lambda*iq;
7     end
8 end
```

if statement should have an else statement, the number of equations, their dimensions and order should remain equal in each branch of '`if-else`' statements, and the equations should contain the same variable in terms of the others.

Once all the necessary sections are defined, the file should be stored in a package directory. The package directory name should start with the '`+`' character. The directory hierarchy defines the library structure in Simscape. The directory's parent directory should be one of the MATLAB paths. This directory should be made the current directory in MATLAB. Executing the '`ssc_build`' command in the command window will generate a library with the custom components. This command has to be executed each time the code in existing components is changed, or new components are added to the library. After the code is built in the manner described, the capacitor block is created as shown in Figure 6. Double-clicking the block opens a dialog box as shown in Figure 7. The block can be dragged into simulations and used similar to standard library blocks.

The Simscape components created in such a manner are called behavioral components, i.e. mathematical equations are used to define such components. Another way of creating components is by using pre-built library components and connecting them using the Simscape language. Such components are called composite components and require additional sections such as '`Structure`' and '`Components`' to specify the component connections, etc. The composite components will not be discussed as

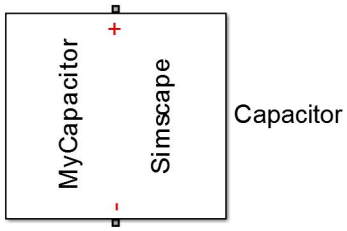

Figure 6: Unconnected capacitor Simscape block.

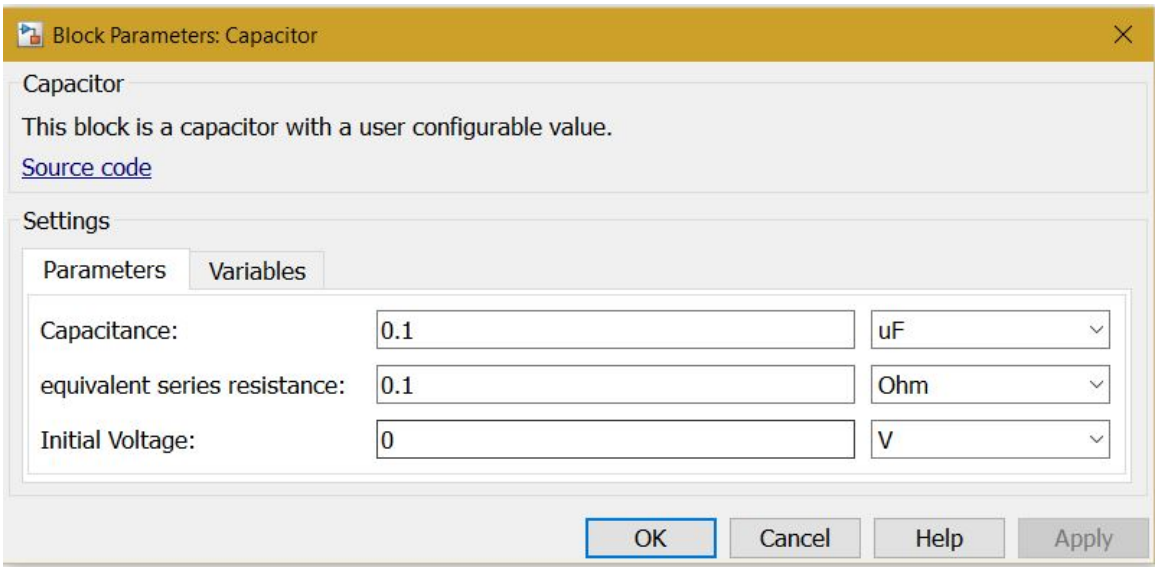

Figure 7: Capacitor block dialog box.

a part of this thesis. The following sections cover the branches and equations sections of the individual components. The complete code for the same can be found in the Appendix.

3.2 Electric Machine

The electric machine is the component where power exchange between the electrical and mechanical domains takes place. As a result, the branches and equations sections of this component consist of electrical, mechanical and electro-mechanical equations. The branches section of the machine block is shown in Listing 4. The first three branches are the electrical branches. These equations state that the currents through the stator flow to the common node 'n'. The fourth branch is the mechanical branch.

Listing 4: Electric machine branches section

```
branches
    ia : a.i -> n.i;
    ib : b.i -> n.i;
    ic : c.i -> n.i;
    T : C.t -> R.t;
end
```

The positive through variable, torque '`T`', flows from node '`C`' to '`R`'.

The equations section of the machine is shown in Listing 5. The '`let-in-end`' construct is used to define two intermediate terms that are used in the equations. The first three equations are used to establish the electrical across variables and the fourth equation establishes the across variable for the mechanical domain. Lines 11 and 12 are mechanical and electro-mechanical equations respectively. The d-axis and q-axis voltages are calculated based on the transformations defined by (4). Lines 18, 19 and 20 are the machine dynamics in the dq system. The currents calculated from

Listing 5: Electric machine equations section

```
1  equations
2      let
3          Lambda = sqrt(3/2)*Lambda_m;
4          Lo=0.1*L;
5      in
6          va == a.v-n.v;
7          vb == b.v-n.v;
8          vc == c.v-n.v;
9          w == R.w-C.w;
10
11         w == theta.der;
12         T == N*Lambda*iq;
13
14         vd == sqrt(2/3)*(va*cos(N*theta) + vb*cos(N*theta
               -(2*pi/3)) + vc*cos(N*theta+(2*pi/3)));
15         vq == sqrt(2/3)*(-va*sin(N*theta) - vb*sin(N*theta
               -(2*pi/3)) - vc*sin(N*theta+(2*pi/3)));
16         vo == sqrt(1/3)*(va + vb + vc);
17
18         L*iq.der == vq-Rs*iq-N*w*(L*id+Lambda);
19         L*id.der == vd-Rs*id+N*w*L*iq;
20         Lo*io.der == vo-Rs*io;
21
22         ia == sqrt(2/3)*(id*cos(N*theta)-iq*sin(N*theta) +(
               io/sqrt(2)));
23         ib == sqrt(2/3)*(id*cos(N*theta-(2*pi/3))-iq*sin(N*
               theta-(2*pi/3)) +(io/sqrt(2)));
24         ic == sqrt(2/3)*(id*cos(N*theta+(2*pi/3))-iq*sin(N*
               theta+(2*pi/3)) +(io/sqrt(2)));
25     end
26 end
```

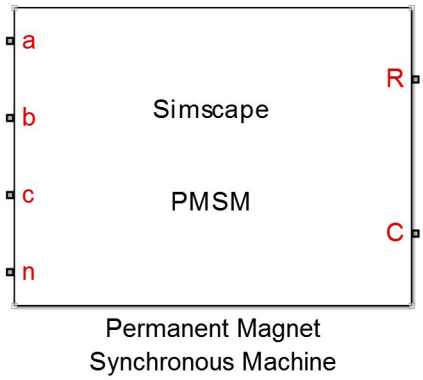

Figure 8: Permanent magnet synchronous machine block, equivalent to Figure 2.

the dynamics are then used as through variables by using (7). The entire equations section thus implements (8), (9) and (11). The zero axis dynamics are considered here for the sake of generality. Building the block creates a block as shown in Figure 8.

3.3 Power Converter

3.3.1 Power Stage

The power stage consists of multiple through and multiple across variables. The branches section of the power stage is given in Listing 6. The node '`n`' acts as the reference for both the DC and AC sides. Multiple through variables require multiple lines in the branches section as shown in the code. The variables '`idc`', '`iA`', '`iB`' and '`iC`' then define the current flow inside the component.

The equations section of the power stage is shown in Listing 7. The lines 2, 6, 8 and 10 establish the across variables for the component. The capacitor dynamic

Listing 6: Power stage branches section

```
1    branches
2        idc : p.i -> n.i;
3        iA : n.i -> a.i;
4        iB : n.i -> b.i;
5        iC : n.i -> c.i;
6    end
```

Listing 7: Power stage equations section

```
1     equations
2         vdc == p.v-n.v;
3         vdc == vcap + icap*ESR;
4         icap == C*vcap.der;
5         vdc*idc == ((vA*iA + vB*iB + vC*iC)) + vdc*icap;
6         vA == a.v - n.v;
7         vA == vA_ref;
8         vB == b.v - n.v;
9         vB == vB_ref;
10        vC == c.v - n.v;
11        vC == vC_ref;
12    end
```

equation, line 4, specifies the current through the DC bus capacitor. The line 3 uses Kirchhoff's Voltage Law to establish the DC side voltage '`vdc`'. As discussed earlier, by controlling the voltages at the AC side of the power converter, the voltages experienced by the motor can be controlled. These voltage commands are received from the power converter controller. In the power stage block, they are programmed as physical input ports. Lines 7, 9 and 11 then impress these voltage commands on the AC side terminals. A power balance on the AC and DC sides is used to calculate the DC side current that enters the three phase bridge. All these equations thus implement (13) through (16). Building the code creates a block as shown in Figure

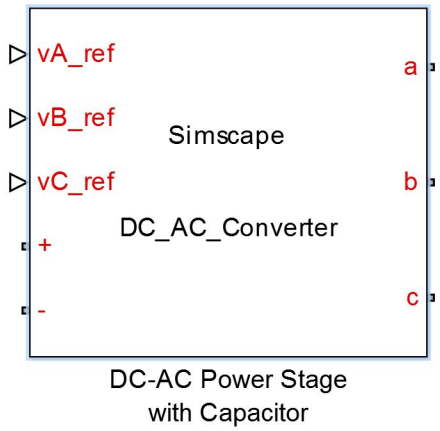

Figure 9: Power stage block, equivalent to Figure 4.

9. This block simulates the operation of the circuit shown in Figure 4.

3.3.2 Power Converter Controller

The power converter controller uses torque request, current feedback, angular velocity feedback and angular position feedback to generate voltage commands. There are no conserving ports required for this block. All the signals and commands are programmed using input and output ports. As a result, there are no domain through variables linked with the block. Therefore, the branches section of the code is not required. There are no domain across variables either. But the equations section of the code is required to generate output commands and solve PI controller differential equations.

The equations section of the code is given in Listing 8. This block requires intermediate terms and hence uses the '`let-in-end`' construct in the equations section.

Listing 8: Power converter controller equations section

```
equations
    let
        id = sqrt(2/3)*(iABC(1)*cos(N*theta) + iABC(2)*cos(N
           *theta-(2*pi/3)) + iABC(3)*cos(N*theta+(2*pi/3))
           );          % idqo(1);
        iq = sqrt(2/3)*(-iABC(1)*sin(N*theta) - iABC(2)*sin(
           N*theta-(2*pi/3)) - iABC(3)*sin(N*theta+(2*pi/3))
            );          % idqo(2);
        id_ref = {0,'A'};
        iq_ref = {sqrt(2/3)*Te_ref/(N*lambda_m), 'A'};
        vd_ref = -kp*(id-id_ref) -ki*xd;
        vq_ref = -kp*(iq-iq_ref) -ki*xq;
    in
        xd.der == (id-id_ref);
        xq.der == (iq-iq_ref);
        vA_ref == sqrt(2/3)*( vd_ref*cos(N*theta) - vq_ref*
           sin(N*theta) );
        vB_ref == sqrt(2/3)*( vd_ref*cos(N*theta-(2*pi/3)) -
            vq_ref*sin(N*theta-(2*pi/3)) );
        vC_ref == sqrt(2/3)*( vd_ref*cos(N*theta+(2*pi/3)) -
            vq_ref*sin(N*theta+(2*pi/3)) );
    end
end
```

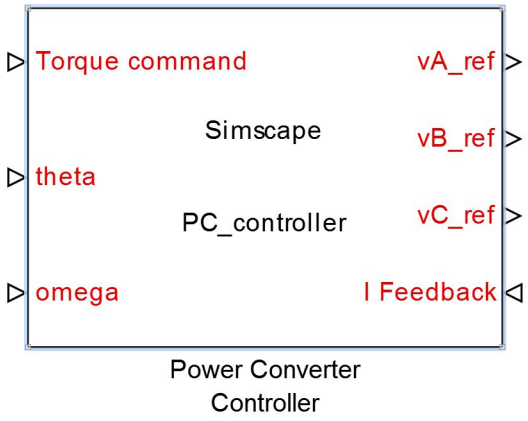

Figure 10: Power converter controller block.

The d-axis and q-axis feedback currents, lines 3 and 4, are calculated from the stator currents (‘`iABC`’) and rotor angular position (‘`theta`’) feedback. The d-axis and q-axis reference currents, lines 5 and 6, are calculated from the desired torque request. Lines 5 shows a different way of assigning units to terms in the equations section. The d-axis and q-axis voltage commands are generated by using PI control on the current error as shown on lines 7 and 8. Lines 10 and 11 are the integrals of the respective current errors. Lines 12, 13 and 14 assign the calculated voltage commands to the output ports. The entire equations section thus simulates (5), (6) and (24) through (27). Building the code generates the block as shown in Figure 10.

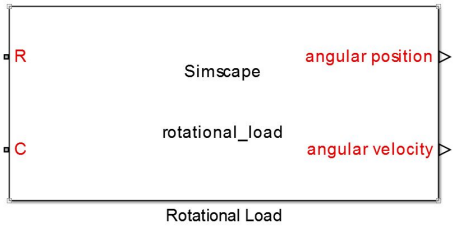

Figure 11: Rotational load block.

Listing 9: Rotational load branches and equations sections

```
1  branches
2      t: R.t -> C.t;
3  end
4  equations
5      w == R.w-C.w;
6      t == J*w.der + B*w;
7      w == theta.der;
8      A == theta;
9      W == w;
10 end
```

3.4 Rotational Load

The branches and equations sections of the rotational load block are given in Listing 9. The only branch for the rotational load implies that the positive through variable, torque '`t`', flows from the node '`R`' to the node '`C`'.

The across variable is established in the first equation. The torque equation as given by (28) is programmed by line 6. The speed dynamics are programmed in line 7 which is given by (29). Lines 8 and 9 output the angular position and angular velocity of the load. This can be fed back to the controller for its calculation of dq variables in the controller.

3.5 DC Power Source

The branches and equations sections of the DC power source are given in Listing 10. The through variable '`i_dc`' flows from node '`p`' to node '`n`'. This inherently assigns a sign convention to the variable. In other words, a positive current draw through

Listing 10: DC power source branches and equations section

```
1      branches
2          i_dc    : p.i -> n.i;
3      end
4      equations
5          v_dc == p.v - n.v;
6          v_dc == v_in - i_dc*R;
7      end
```

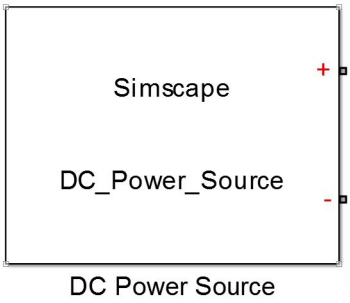

Figure 12: DC power source block, equivalent to Figure 5.

node '`p`', outside the DC power source would mean negative '`i_dc`' flowing inside the component.

The first equation is used to establish the across variable in the component. This variable is a combination of the voltage source value and the voltage drop across the internal series resistance. Thus, the combination of these two equations implements (30). The equivalent circuit is seen in Figure 5. Figure 12 shows the component block after the code is built.

3.6 Resistor

The branches and equations sections of the resistor block are given in Listing 11. The branches section establishes the flow of the through variable through the resistor. The first equation establishes the across variable for the resistor block. The second equation simulates Ohm's Law. The block shown in Figure 13 is the output after

Listing 11: Resistor branches and equations section

```
1     branches
2         idc    : p.i -> n.i;
3     end
4
5     equations
6         vdc == p.v - n.v;
7         vdc == idc * r;
8     end
```

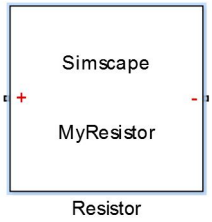

Figure 13: Resistor block.

building the code given.

3.7 Constructing the Drive System

3.7.1 System Overview

Building all the components will create a Simscape library. The components should be connected as shown in Figure 14. This figure depicts the torque control drive system of a permanent magnet synchronous machine. The component parameter values simulated are covered below. The component interconnections are also discussed in brief.

The DC power source is connected across R_1 and in series with R_2. The node '+' of both the resistors and DC power source are connected together. The node '−' of the DC power source and the load resistor are connected together to the reference and the node '−' of the DC side of the power stage of the converter. The node '−' of the pre-charge resistor is connected to the node '+' of the DC side of the power stage of the converter. The values of the parameters set in the two resistors and the DC power source are as given in Table 1, where the script symbol represents the name

Table 1: DC Power Source and Resistor Parameter Values

Parameter	Math Equation Symbol	Script Symbol	Value	Unit
Input Voltage	V_{dc}	V_dc	80	V
Series Resistance	R_{source}	R_source	50×10^{-3}	Ω
Resistance	R_1	R_load	8	Ω
Resistance	R_2	R_interface	35	Ω

of the parameter in the initialization script file given at the end of this section. The different components are separated by horizontal lines in the table. Here, '`R_load`' is

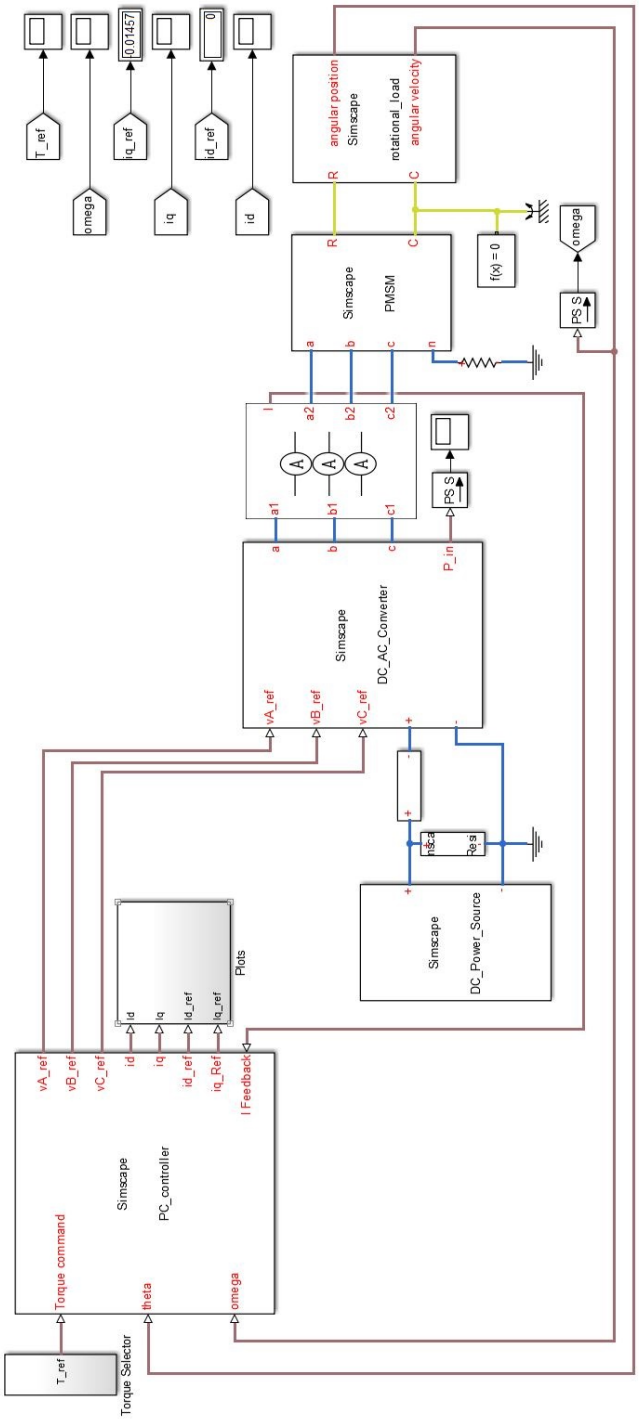

Figure 14: Connections of the drive system, equivalent to Figure 1.

same as R_1 and '`R_interface`' is same as R_2 from Figure 1.

The power stage of the converter receives the commands from the power converter controller and the command signals are shown by brown wires coming into the '`DC_AC_Converter`' block in Figure 14. The AC side of the power stage is connected to the current sensor. The current sensor provides stator current feedback to the power converter controller. The parameters for configuring the power stage block are given in Table 2.

Table 2: Power Stage with Capacitor Parameter Values

Parameter	Math Equation Symbol	Script Symbol	Value	Unit
Capacitance	C_{cap}	C	1880×10^{-6}	F
Equivalent Series Resistance	R_{ESR}	ESR	50×10^{-3}	Ω
Initial Voltage	V_0	V_cap	80	V

The '`PMSM`' block electrical side is connected to the other side of the current sensor. The mechanical side of the '`PMSM`' is connected to the '`rotational_load`'. The '`C`' nodes of the load and the machine are connected to the mechanical reference. The '`R`' nodes of both the components are connected directly to each other. The parameters for the machine configuration are given in Table 3.

Table 3: Machine Parameter Values

Parameter	Math Equation Symbol	Script Symbol	Value	Unit
Number of Pole Pairs	N	N	6	1
Resistance per Phase	R_s	R	9.26×10^{-3}	Ω
Inductance per Phase	L_s	L	0.137×10^{-3}	H
Permanent Magnet Flux	Λ_m	Lambda_m	59.5×10^{-3}	Wb

The load block provides the angular position and velocity feedback to the power converter controller. The machine is operated without load in the simulations shown in the next chapter. The parameters for the configuration of the load block, i.e. the

rotor inertia and damping coefficient, are given in Table 4. Since the rotor inertia and damping coefficient are not included in the machine block, they must be accounted for in the rotational load block.

Table 4: Unloaded Rotor Parameter Values

Parameter	Math Equation Symbol	Script Symbol	Value	Unit
Inertia	J	J	28.25×10^{-3}	kg-m^2
Damping Coefficient	B	B	2.45×10^{-3}	N-m/(rad/s)
Initial Offset	$\theta(0)$	-	0	rad

The power converter controller receives feedback from the current sensor and load block. The internal calculations then generate the voltage reference commands for the power stage. The controller receives the torque request from a logic. The logic operates on a parameter `mode`. If `mode` is 1, a sinusoidal torque request is generated with an amplitude of 1 N-m and frequency of 0.5 Hz. If the `mode` is 2, a torque request function, in Nm, is given by

$$T_{ref} = \begin{cases} 1, & (t\%4) = 0 \\ -1, & (t\%4) = 2 \\ 0, & \text{otherwise} \end{cases} \tag{32}$$

where t represents the current simulation time in seconds, and the symbol % represents the modulus operator. The parameters for configuring the '`PC_controller`' block are given in Table 5.

Table 5: Power Converter Controller Parameter Values

Parameter	Math Equation Symbol	Script Symbol	Value	Unit
Number of Pole Pairs	N	N	6	1
Permanent Magnet Flux	Λ_m	Lambda_m	59.5×10^{-3}	Wb
PI tuning parameter	k_p	kp	0.4017	Ω
PI tuning parameter	k_i	ki	308.25	Ω/s

The PI tuning parameters are calculated using pole placement method. The units are assigned to these constants to satisfy unit requirements of the equations given by (26) and (27). The results for the test modes 1 and 2 without load are presented in the following chapter.

All the parameters described above can be set using a script file as shown in Listing 12.

Listing 12: Parameter initialization code

```
clc, clear all, close all

mode = 1;

% Voltage
V_dc = 80;
R_source = 0.05;

% Mechanical parameters
J = 0.02825;
B = 0.00245;

% Resistance parameters
R_interface = 35;
R_load = 8;

% Motor model assigned parameters (Parker 210-150P)
N = 6;
Lambda_m = 59.5e-3;
L = 0.137e-3;
R = 9.26e-3;

% Inverter model assigned parameters (Sevcon G4-S8)
C = 1880e-6;
ESR = 50e-3;
V_cap = 0;

% PI regulator gains
BW = 1500;
kp = 2*L*BW-R;
ki = L*BW^2;
```

3.7.2 Simscape Simulation Execution and Solver Configuration

Simscape simulation execution is similar to Simulink simulation execution. The Simscape simulation follows a sequence, which is described as follows. The simulation starts with *Model Validation*. In this step, the Simscape solver validates the physical model. It checks whether all conserving nodes are connected, if all the parameters are entered correctly, and if the solver configuration block is present. The next step is *Network Construction*, where the across variable and through variable values are propagated. The across variable is the same for the branches connected to a node. The through variable is divided amongst all the branches connected to a node. Based on the parameter values entered and the network configuration, the *Equation Construction* step constructs dynamic and algebraic equations. Dynamic equations are the time differential equations, whereas algebraic equations do not include the time derivatives. After constructing all the equations, only the required variables are retained and others are eliminated. The required variables are mapped to the Simulink state vector for the model.

The next step is *Initial Conditions Computation*. Here the initial conditions are computed for all the variables at time 0 seconds. If some variables are assigned as high priority, the solver assigns the desired values to these variables first and solves the equations for the low priority variable values. If the solver is unable to find values that are consistent with the equations, it relaxes the priority high to low and solves the equations to generate suitable initial conditions. The *Transient Initialization* step fixes all dynamic variables and solves for algebraic variables and derivatives of dynamic variables after the simulation reaches a discontinuity. This step provides a consistent set of initial conditions for the next step, the *Transient Solve*. In this final step, the solver solves the system of equations. The continuous differential equations are integrated to compute variables as functions of time. The solver performs transient solve until the end of simulation time or until it reaches a

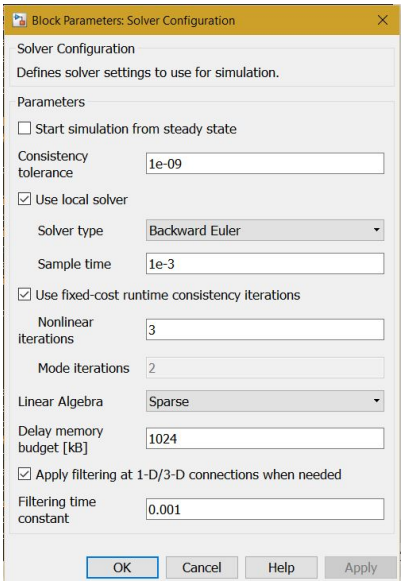

Figure 15: Solver configuration block settings.

discontinuity. If a discontinuity is reached, it cycles back to transient initialization and then attempts to solve again.

The solver configuration block is connected to the model shown in Figure 14. The block provides the solver configuration options. The *Start from steady state* check box commands the solver to start from a steady state. If checked, the solver tries to find a steady state where the outputs of the system do not change for a given constant set of inputs. If such a state exists, the solver starts the variables using these values. The *Consistency tolerance* is used to control the accuracy of the variables in the initial conditions computation or the transient initialization. Increase the number to relax the tolerance and vice versa. The option *Use local solver* converts the model into a discrete model with sample time defined by the *Sample time* parameter.

The *Use fixed-cost runtime consistency iterations* option, when checked, ignores the error if the solver fails to converge on a solution in the transient initialization step within the number of iterations defined by *Non-linear iterations* and *Mode iterations*. This option requires a global fixed step solver. The Simulink solver can then be changed to a discrete solver such as `ode4` and the step size can be controlled to increase

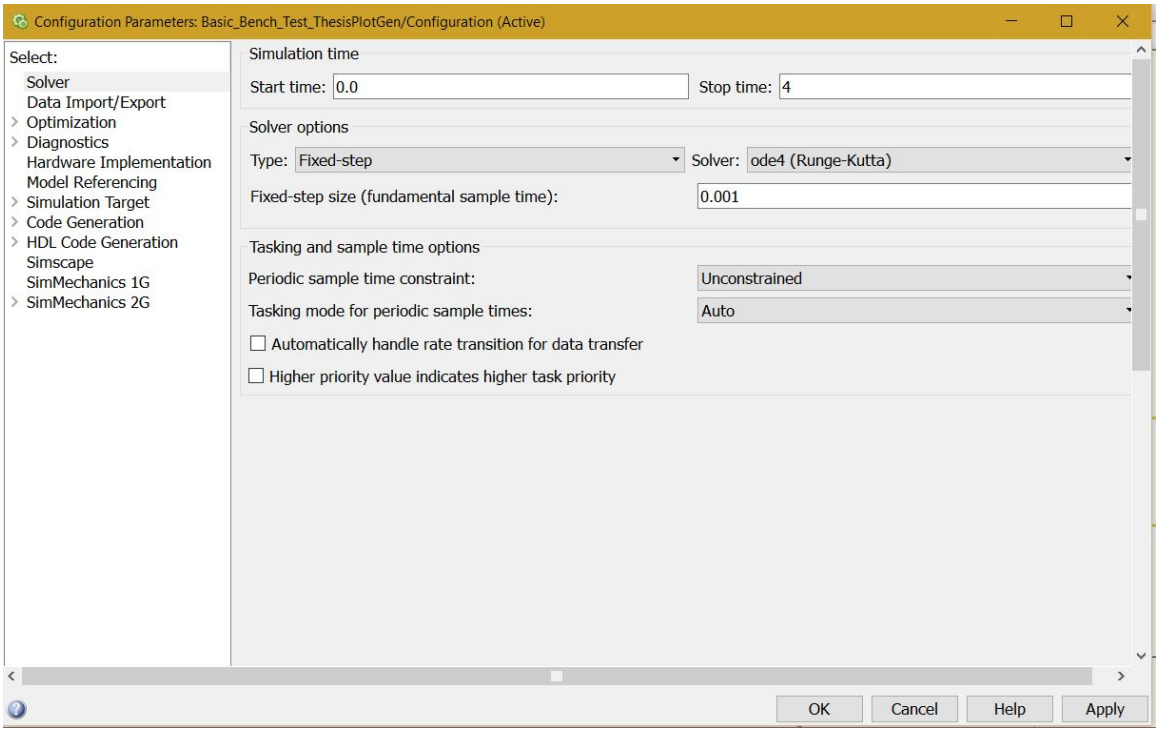

Figure 16: Configuration parameters dialog box settings.

or decrease the detail of the simulation. *Mode iterations* option is not required if using a local solver. The *Linear Algebra* option is used for memory management, Sparse being the more efficient one. Since MATLAB stores each element of the matrix in the same memory space, storing matrix elements of zero value is a wastage of memory and so sparse matrices store only the non-zero values with indices. The *Delay memory budget* is used in case delays are present in the system. The final option is not applicable to the system considered here. The configured solver looks as shown in Figure 15.

Another setting is required in the configuration parameters dialog box. The solver can be changed to fixed step solver to eliminate the step-size related errors and to use the *fixed-cost runtime consistency iterations* option. The solver chosen here is the `ode4` or the fourth order Runge-Kutta solver. The step size used is 1 millisecond. The simulation is able to run on higher as well as lower step sizes. It becomes a matter of accuracy and detail desired from the simulation to decide on the step size. The

configuration parameters dialog box is shown in Figure 16.

CHAPTER IV

RESULTS AND CONCLUSIONS

4.1 Results

The machine block was tested to verify the torque and voltage constants against values provided in the datasheet. The machine block was subjected to the blocked rotor test to verify the torque constant, where the stator was supplied with a known current and the torque produced at the blocked rotor was measured. The machine was also subjected to the open circuit test, where the rotor was supplied with a known angular velocity and voltage induced at the stator was measured to estimate the voltage constant.

The parameter values for the Parker GVM210-150P6 machine were entered in the block, given in Table 3, and the tests were conducted. The machine block was supplied with 100 Arms for the blocked rotor test. The torque produced by the machine block was measured as 75.73 Nm. This was used to calculate the torque constant as 0.7573 Nm/Arms, and the datasheet value for the same is given as 0.757 Nm/Arms. In another test, the rotor was connected to a speed source of 100 rad/s. The peak voltage induced at the stator was measured as 61.83 V. This results in the value of voltage constant as 0.6183 V/(rad/s). The datasheet value for the voltage constant is given as 0.6184 V/(rad/s). From both these tests, it can be concluded that the machine model is a close match to the actual physical machine.

Once the machine block test results were satisfactory, bench test simulations were performed to check if the machine was able to produce requested torque. The system shown in Figure 14 was simulated using torque request modes 1 and 2, discussed in Section 3.7.1, without load. The configuration parameters were set to the values as

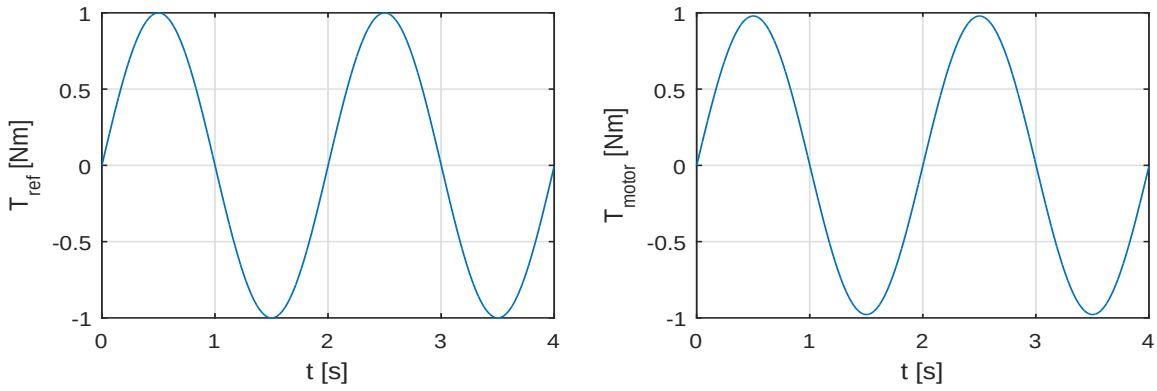

Figure 17: Comparison of requested and produced torque.

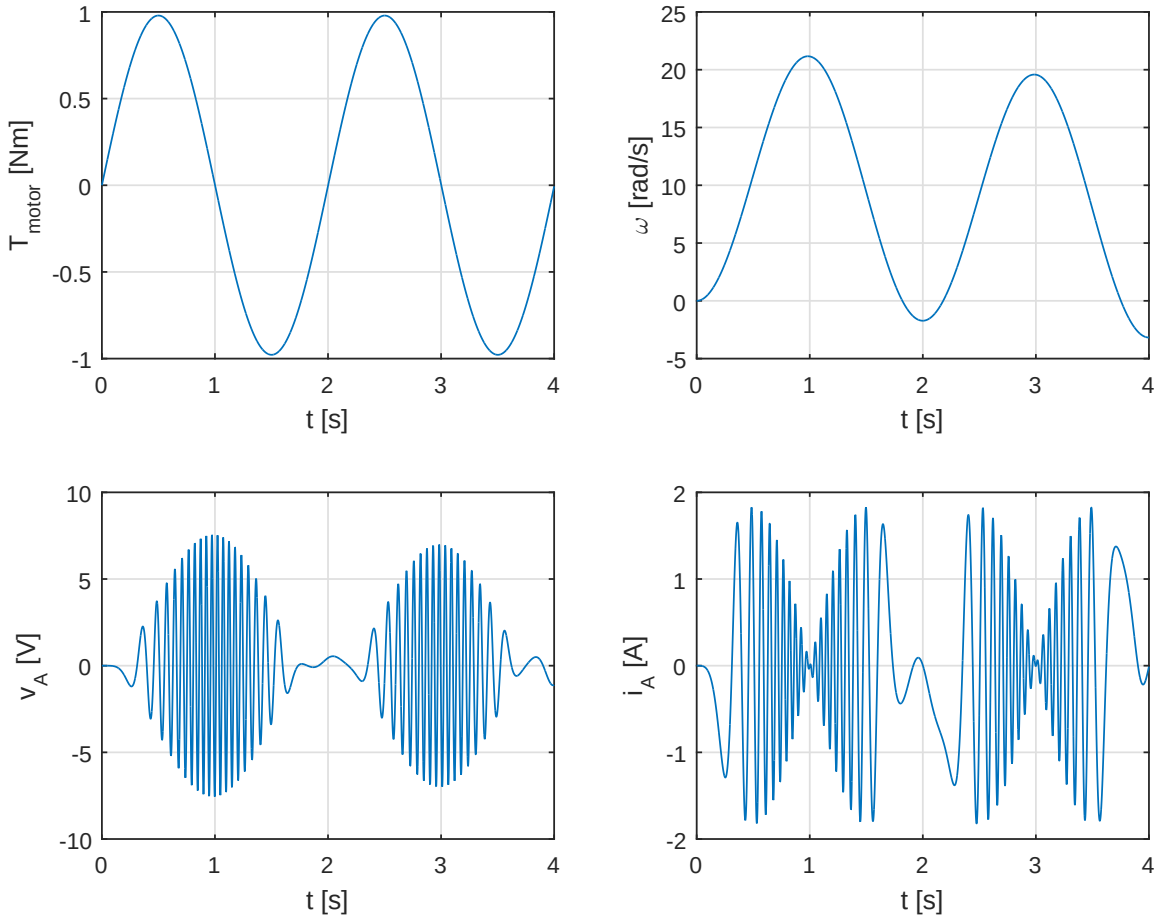

Figure 18: Unloaded rotor with sinusoidal torque test results.

described in Section 3.7.1. The only parameter that changed was mode.

The first case was mode 1 without load. As seen from Figure 17, the torque output of the machine matches the torque request given to the drive system. Figure

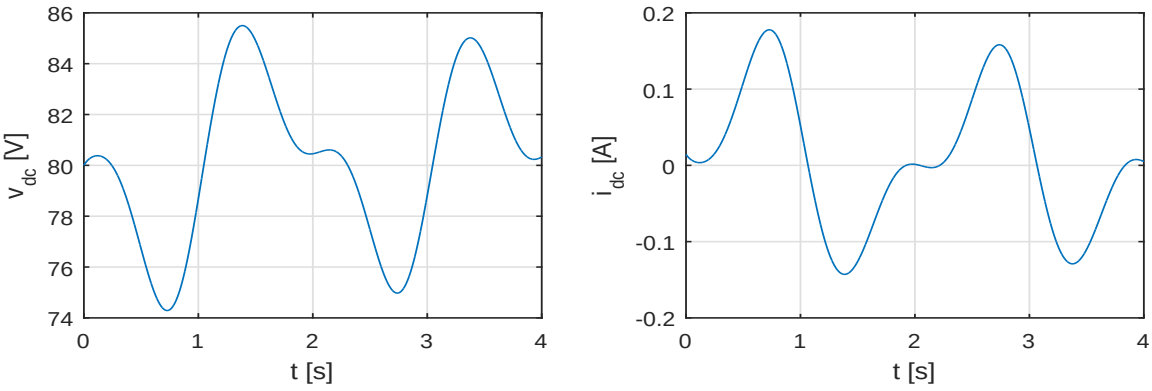

Figure 19: DC side voltage and current for unloaded rotor with sinusoidal torque.

18 shows the motor torque, angular velocity, A-phase voltage at power converter AC side and stator A-phase current. The frequency of the AC side voltage and current increases with speed which follows from (1), (2) and (3). The amplitude of the current varies proportionally with the torque. The voltage and current on the DC side of the power converter are as shown in Figure 19. The DC side current goes negative when the machine is regenerating, which is seen in Figure 19. As the system is operated without load, the equilibrium speed reached for a given torque command was relatively high. The final value theorem estimates the speed for the rotor, only considering rotor inertia and damping, to be approximately 408.16 rad/s for a step input of 1 Nm torque, and the time constant for such a system is approximately 11.55 seconds. Assuming the system takes about five time constants to reach steady state, the rotor would require about 57.75 seconds to reach steady state. Since the torque request here is sinusoidal, the angular velocity also follows a sinusoidal pattern. Note that the peak of the angular velocity is decreasing as seen in Figure 18. This is because the system takes about 58 seconds to reach steady state at which point the speed is symmetric about the time axis and the peak will continue to be constant thereafter.

In mode 2, (32) generates the torque request. The request and the actual torque produced for the unloaded rotor are shown in Figure 21. As discussed earlier, the

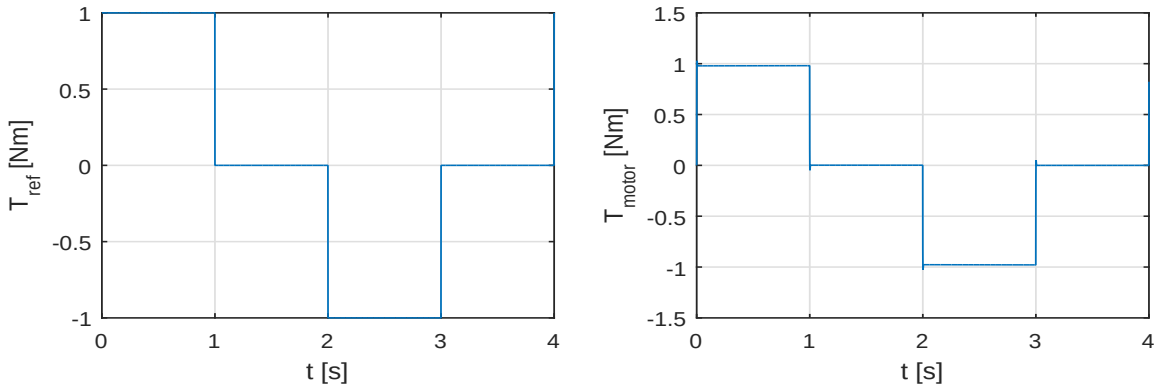

Figure 20: Comparison of requested and produced torque.

final value for the angular velocity and the settling time for it are about 408.16 rad/s and 57.75 seconds respectively, the speed of the rotor after 1 second of constant torque therefore simply seems linear. The AC side voltage and current frequencies

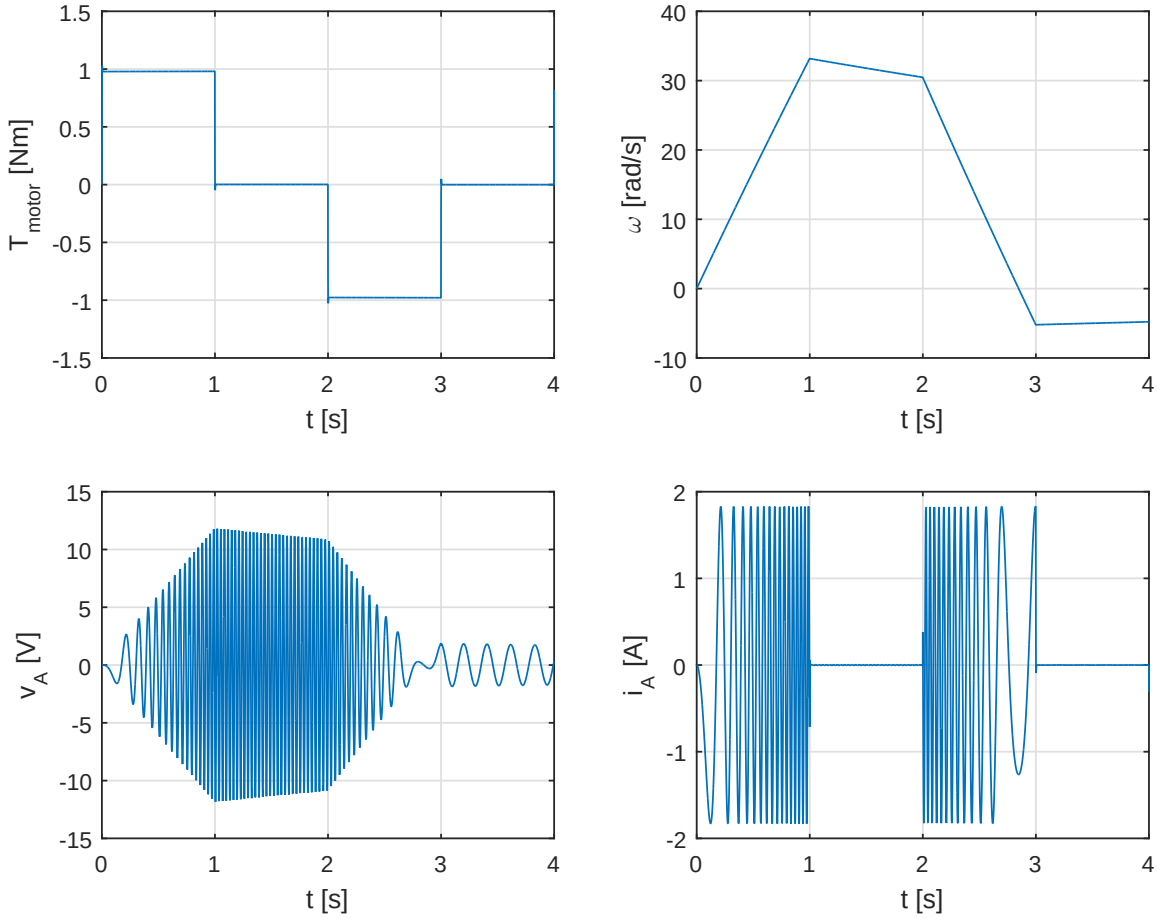

Figure 21: Unloaded rotor with square torque test results.

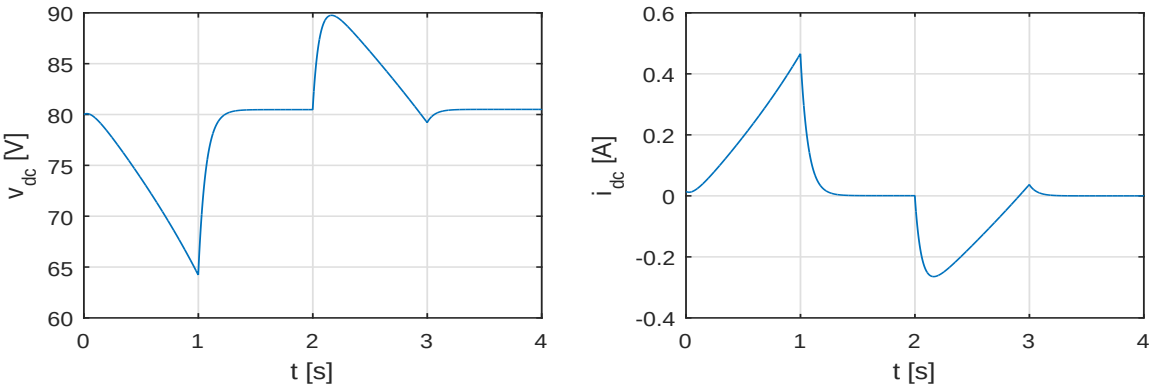

Figure 22: DC side voltage and current for unloaded rotor with square torque.

also increase with increasing angular velocity. The AC side current is zero when the torque request is zero. The DC side voltage of the power converter exhibits first order dynamics due to the presence of the capacitor. The DC side current therefore goes to zero only once the voltage across the capacitor reaches 80 V. During the first second, on the mechanical side, a positive torque and a positive angular velocity results in a positive power out of the machine, which is given by the positive input power, i.e. the positive DC side voltage and positive DC side current. During the third second, the negative torque and positive angular velocity gives a negative power and hence the machine is operating as a generator as can be seen by the DC side current going negative in Figure 22. At the end of the third second, the angular velocity becomes slightly negative causing power output and input to become positive as seen by the small positive bump in the DC side current at time 3 seconds.

4.2 Conclusions

The Simscape language creates an opportunity for engineers to simulate any complex system that cannot be built using Foundation Library components, or if the user analysis differs from the Simscape block analysis, or if the user is not comfortable enough with the equations executed in the standard blocks. The systems can be further tuned by adding functionality to the blocks such as efficiency, power losses,

thermal management, etc., for simulating systems that more closely resemble the physical world. The differentiation between the various domains limits the scope of mistakes that may occur in Simulink simulations. This approach also helps in making the simulations more readable by reducing the number of blocks and wires. Clever thinking and analysis can further help improve the speed of execution of the simulations.

With regard to the system implemented using Simscape language in this thesis, the behavior of the entire drive system in simulation was as predicted. The simulation presented here runs with a fixed step solver with a step size of 1 millisecond and is able to simulate more than 4 seconds of system operation within one second of real time. This computational efficiency is important because the huge amount of time taken for simulation is usually a problem with simulation tools.

4.3 Possible Future Work

The purpose of this thesis is to serve as a guide for creating custom components using Simscape language. The work presented here can be further enhanced. The future work on this project could entail inclusion of control in the field weakening region. The motor block may include an option to select the winding type and the type of dq transformation. The motor block could also incorporate the angular speed and position sensor that is available in commercial off-the-shelf products. The rotor inertia and damping coefficients can be added to the motor block. The pre-charge circuit and controller can be included in the battery block so as to closely resemble the actual physical system. The thermal analysis and equations for each component may be included in each respective block along with the cooling circuits.

APPENDIX A

SIMSCAPE CODE

A.1 Electric Machine

```
component PMSM
% Permanent Magnet Synchronous Machine
% This block simulates a Y-connected permanent magnet
   synchronous machine.
% Note: L0 considered as 0.1*Ld.

parameters
    N = {6,'1'};                        % Number of Pole
       Pairs
    Rs = {9.26e-3,'Ohm'};               % Series Resistance
       per Phase
    L = {0.137e-3,'H'};                 % d and q axis
       Inductance
    Lambda_m = {59.5e-3,'Wb'};          % Permanent Magnet
       Flux Linkage
end

nodes
    a = foundation.electrical.electrical;               % a:
       left
    b = foundation.electrical.electrical;               % b:
       left
```

```
16      c = foundation.electrical.electrical;                % c:
            left
17      n = foundation.electrical.electrical;                % n:
            left
18      R = foundation.mechanical.rotational.rotational;     % R:
            right
19      C = foundation.mechanical.rotational.rotational;     % C:
            right
20  end
21
22  variables
23      va = {0,'V'};                          % Phase A Voltage
24      vb = {0,'V'};                          % Phase B Voltage
25      vc = {0,'V'};                          % Phase C Voltage
26      w = {0,'rad/s'};                       % Angular Speed
27      ia = {0,'A'};                          % Phase A Current
28      ib = {0,'A'};                          % Phase B Current
29      ic = {0,'A'};                          % Phase C Current
30      T = {0,'N*m'};                         % Torque
31      id = {0,'A'};                          % d-axis Current
32      iq = {0,'A'};                          % q-axis Current
33      vd = {0,'V'};                          % d-axis Voltage
34      vq = {0,'V'};                          % q-axis Voltage
35      theta = {0,'rad'};                     % Angular Position
36  end
37
38  variables(Access = protected)
39      vo = { 0, 'V' };                       % o-axis Voltage
40      io = { 0, 'A' };                       % o-axis Current
```

```
41 end
42
43 branches
44     ia : a.i -> n.i;
45     ib : b.i -> n.i;
46     ic : c.i -> n.i;
47     T : C.t -> R.t;
48 end
49
50 equations
51     let
52         Lambda = sqrt(3/2)*Lambda_m;
53         Lo=0.1*L;
54     in
55         va == a.v-n.v;
56         vb == b.v-n.v;
57         vc == c.v-n.v;
58         w == R.w-C.w;
59
60         w == theta.der;
61         T == N*Lambda*iq;
62
63         vd == sqrt(2/3)*(va*cos(N*theta) + vb*cos(N*theta
               -(2*pi/3)) + vc*cos(N*theta+(2*pi/3)));
64         vq == sqrt(2/3)*(-va*sin(N*theta) - vb*sin(N*theta
               -(2*pi/3)) - vc*sin(N*theta+(2*pi/3)));
65         vo == sqrt(1/3)*(va + vb + vc);
66
67         L*iq.der == vq-Rs*iq-N*w*(L*id+Lambda);
```

```
68          L*id.der == vd-Rs*id+N*w*L*iq;
69          Lo*io.der == vo-Rs*io;
70
71          ia == sqrt(2/3)*(id*cos(N*theta)-iq*sin(N*theta) +(
               io/sqrt(2)));
72          ib == sqrt(2/3)*(id*cos(N*theta-(2*pi/3))-iq*sin(N*
               theta-(2*pi/3)) +(io/sqrt(2)));
73          ic == sqrt(2/3)*(id*cos(N*theta+(2*pi/3))-iq*sin(N*
               theta+(2*pi/3)) +(io/sqrt(2)));
74      end
75  end
76
77  end
```

A.2 Power Converter

A.2.1 Power Stage

```
component DC_AC_Converter
% DC-AC Power Stage with Capacitor
% This block simulates the DC-AC power conversion and vice-
   versa. The control strategy needs to generate vA_ref,
% vB_ref and vC_ref as physical signals and the converter
   will generate appropriate voltages at the corresponding
   nodes.
% This block includes an internal DC-bus capacitor.

    parameters
        C = {10e-6, 'F'};          % Bus capacitance
        ESR = {0.001, 'Ohm'};      % Equivalent series
           resistance of bus capacitor
        v0 = {0, 'V'};             % Initial Capacitor
           Voltage
    end

    inputs
        vA_ref = { 0.0, 'V' }; % vA_ref:left
        vB_ref = { 0.0, 'V' }; % vB_ref:left
        vC_ref = { 0.0, 'V' }; % vC_ref:left
    end

    nodes
        p = foundation.electrical.electrical; % +:left
        n = foundation.electrical.electrical; % -:left
```

```
22          a = foundation.electrical.electrical; % a:right
23          b = foundation.electrical.electrical; % b:right
24          c = foundation.electrical.electrical; % c:right
25      end
26
27      variables
28          iA = { 0, 'A' };     % A Phase Current
29          vA = { 0, 'V' };     % A Phase Voltage
30          iB = { 0, 'A' };     % B Phase Current
31          vB = { 0, 'V' };     % B Phase Voltage
32          iC = { 0, 'A' };     % C Phase Current
33          vC = { 0, 'V' };     % C Phase Voltage
34          idc = { 0, 'A' };    % DC-side Current
35          vdc = {0, 'V' };;    % DC-side Voltage
36          vcap = {value={0,'V'},priority=priority.high};   %
               Bus Capacitance Voltage
37          icap = {0, 'A' };    % Bus Capacitance Current
38      end
39
40
41      function setup
42      if C<=0
43          pm_error('simscape:GreaterThanOrEqualToZero','Bus
               capacitance')
44      end
45      if ESR<=0
46          pm_error('simscape:GreaterThanOrEqualToZero','
               Equivalent series resistance of bus capacitor')
47      end
```

```
48      if v0<0
49          pm_error('simscape:GreaterThanZero','Initial
                Capacitor Voltage')
50      end
51      vcap = v0;
52      end
53
54      branches
55          idc : p.i -> n.i;
56          iA : n.i -> a.i;
57          iB : n.i -> b.i;
58          iC : n.i -> c.i;
59      end
60
61      equations
62          vdc == p.v-n.v;
63          vdc == vcap + icap*ESR;
64          icap == C*vcap.der;
65          vdc*idc == ((vA*iA + vB*iB + vC*iC)) + vdc*icap;
66          vA == a.v - n.v;
67          vA == vA_ref;
68          vB == b.v - n.v;
69          vB == vB_ref;
70          vC == c.v - n.v;
71          vC == vC_ref;
72      end
73  end
```

A.2.2 Power Converter Controller

```
component PC_controller
% Power Converter Controller
% This block generates voltage commands for the DC to AC
   power stage and uses MTPA algorithm.

    parameters
        N = {6, '1'};                    % Number of rotor
           pole pairs of motor
        lambda_m = {59.5e-3,'Wb'};       % Permanent magnet
           flux linkages of rotor
        kp = {0,'Ohm'};                  % PI tuning
           parameter: kp
        ki = {0,'Ohm/s'};                % PI tuning
           parameter: ki
    end

    outputs
        vA_ref = { 0.0, 'V' };       % vA_ref:right
        vB_ref = { 0.0, 'V' };       % vB_ref:right
        vC_ref = { 0.0, 'V' };       % vC_ref:right
    end

    inputs
        Te_ref = {0, 'N*m'};    % Torque command: left
        theta = {0, 'rad'};     % theta:left
        omega = {0, 'rad/s'};   % omega:left
        iABC = {[0,0,0],'A'};   % I Feedback:right
```

```
23      end
24
25      variables(Access = protected)
26          xd = {value={0,'A*s'},priority=priority.high};      %
                Integral error for d-axis
27          xq = {value={0,'A*s'},priority=priority.high};      %
                Integral error for q-axis
28      end
29
30  equations
31      let
32          id = sqrt(2/3)*(iABC(1)*cos(N*theta) + iABC(2)*cos(N
                *theta-(2*pi/3)) + iABC(3)*cos(N*theta+(2*pi/3))
                );          % idqo(1);
33          iq = sqrt(2/3)*(-iABC(1)*sin(N*theta) - iABC(2)*sin(
                N*theta-(2*pi/3)) - iABC(3)*sin(N*theta+(2*pi/3))
                 );          % idqo(2);
34          id_ref = {0,'A'};
35          iq_ref = {sqrt(2/3)*Te_ref/(N*lambda_m), 'A'};
36          vd_ref = -kp*(id-id_ref) -ki*xd;
37          vq_ref = -kp*(iq-iq_ref) -ki*xq;
38      in
39          xd.der == (id-id_ref);
40          xq.der == (iq-iq_ref);
41          vA_ref == sqrt(2/3)*( vd_ref*cos(N*theta) - vq_ref*
                sin(N*theta) );
42          vB_ref == sqrt(2/3)*( vd_ref*cos(N*theta-(2*pi/3)) -
                 vq_ref*sin(N*theta-(2*pi/3)) );
```

```
43          vC_ref == sqrt(2/3)*( vd_ref*cos(N*theta+(2*pi/3)) -
                vq_ref*sin(N*theta+(2*pi/3)) );
44      end
45  end
46  end
```

A.3 DC Power Source

```
component DC_Power_Source
% DC Power Source
% This block represents a regulated DC source with a series
   resistance.

    parameters
        v_in = {10,'V'}; % Input voltage
        R = {10,'Ohm'}; % Series resistance
    end

    nodes
        p = foundation.electrical.electrical; % +:right
        n = foundation.electrical.electrical; % -:right
    end

    variables
        i_dc = { 0, 'A' }; % DC-side Current
        v_dc = { 0, 'V' }; % DC-side Voltage
    end

    function setup
    if v_in<=0
        pm_error('simscape:GreaterThanOrEqualToZero','Input
           Voltage')
    end
    if R <= 0
```

```
25          pm_error('simscape:GreaterThanZero','Series
                resistance')
26      end
27      end
28
29      branches
30          i_dc    : p.i -> n.i;
31      end
32
33      equations
34          v_dc == p.v - n.v;
35          v_dc == v_in - i_dc*R;
36      end
37  end
```

A.4 Rotational Load

```
component rotational_load
% Rotational Load
% This block simulates a rotational load with inertia and
   friction.
% It simulates the equation T=J*w(dot) + B*w where J is the
   inertia and B is the coefficient of friction.
nodes
    R = foundation.mechanical.rotational.rotational; % R:
       left
    C = foundation.mechanical.rotational.rotational; % C:
       left
end

parameters
    J = {0.0283,'kg*m^2'};                % Inertia
    B = {2.6836e-5,'(N*m)*s/rad'};           % Damping
       Coefficient
    init_pos = {0,'rad'};            % Initial position/
       Offset
end

outputs
    A = {0,'rad'};       % angular position
    W = {0,'rad/s'};     % angular velocity
end

variables(Access = protected)
```

```
22      w = {0,'rad/s'};          % angular velocity
23      t = {0,'N*m'};            % torque
24      theta = {0,'rad'};        % angular position
25  end
26
27  function setup
28      if J <= 0
29          pm_error('simscape:GreaterThanOrEqualToZero','
                Inertia')
30      end
31      if B <= 0
32          pm_error('simscape:GreaterThanOrEqualToZero','
                Coefficient of friction')
33      end
34      theta.value = init_pos;
35      theta.priority = priority.high;
36  end
37
38  branches
39      t: R.t -> C.t;
40  end
41  equations
42      w == R.w-C.w;
43      t == J*w.der + B*w;
44      w == theta.der;
45      A == theta;
46      W == w;
47  end
48  end
```

A.5 Resistor

```
component MyResistor
% Resistor
% This block simulates a resistor with a user configurable
   value.
    parameters
        r = {1,'Ohm'}; % Resistance
    end

    nodes
        p = foundation.electrical.electrical; % +:left
        n = foundation.electrical.electrical; % -:right
    end

    variables
        idc = { 0, 'A' }; % DC-side Current
        vdc = { 0, 'V' }; % DC-side Voltage
    end

    branches
        idc   : p.i -> n.i;
    end

    equations
        vdc == p.v - n.v;
        vdc == idc * r;
    end
end
```

REFERENCES

[1] Hassell, T. J., Weaver, W. W., and Oliveira, A. M., "Using matlab's simscape modeling environment as a simulation tool in power electronics and electrical machines courses," IEEE Frontiers in Education Conference (FIE), 2013.

[2] Hyde, R. A., "Using simscape to support system-level design," UKACC International Conference on Control, September 2010.

[3] Li, C., "Development of simscape simulation model for power system stability analysis," (Atlantis, Bahamas), Asia-Pacific Power and Energy Engineering Conference, March 2012.

[4] Taylor, D., *ECE 4550 Lecture Notes.*

[5] The MathWorks, I., *Simscape Language Guide.* The MathWorks, Inc.